lexiDB: A Scalable Corpus Database Management System

Matthew Coole, Paul Rayson and John Mariani

***Abstract*—lexiDB is a scalable corpus database management system designed to fulfill corpus linguistics retrieval queries on multi-billion-word multiply-annotated corpora. It is based on a distributed architecture that allows the system to scale out to support ever larger text collections. This paper presents an overview of the architecture behind lexiDB as well as a demonstration of its functionality. We present lexiDB's performance metrics based on the AWS (Amazon Web Services) infrastructure with two part-of-speech and semantically tagged billion word corpora: Historical Hansard and EEBO (Early English Books Online).**

I. Introduction

Corpora utilised by corpus linguists have steadily grown in scale and complexity over the last fifty years. Beginning with relatively small corpora (although they were considered large at the time) of one million words such as Brown [5] in the 1960s, the size of corpora has been increasing by an order of magnitude roughly every 10 years. In the 1990s the British National Corpus (BNC)[1] was created with one hundred million words and now corpora of interest to linguists order in the billions of words with Historical Hansard[2] and Early English Books Online (EEBO)[3] being prime examples. In parallel to this growth in size of the raw text used in corpora so too has there been an increase in the number of levels of annotation attached to such corpora. Beginning with simple part-of-speech (POS) tagging and lemmatisation, linguists now utilise more advanced annotation such as dependency parsing, semantic tags and historical spelling variants when conducting corpus analysis. This motivates the need for retrieval software and tools that are capable of supporting annotated corpus data at this scale and complexity.

In big data terms, increasing the 'volume' of corpora provides greater numbers of examples for mid to low frequency words and linguistic features which is important for analysis purposes. The 'variety' of data included within a corpus is also important to achieve for improved representativeness and coverage of the types of language being studied. In this paper, we address issues of 'velocity' (the application of parallel or distributed methods), consideration of which is vitally important since the current crop of corpus linguistics retrieval tools are struggling to cope with the ever increasing scale of corpora.

Typically corpus linguists rely on five main retrieval methods in order to perform their analysis: concordances, collocations, clusters (n-grams), keyword lists and frequency lists. Whilst other more complex forms of analysis exist they often are built on top of one or more of these basic methods or are sometimes subtle variations of such queries. These query types are generally not fully or efficiently supported by traditional DBMSs (Database Management Systems) or IR (Information Retrieval) systems as shown in previous work [1]. Some systems have limited support for keyword in context search (concordances) but in order to support these query types fully, corpus linguists must usually rely on a tool built on top of an existing retrieval or database system.

Software that can be used locally on desktop PCs are sometimes favored by linguists as they allow them the flexibility to use their own corpora and to perform analysis without reliance on anything more than a laptop. Tools such as WordSmith[4] and AntConc[5] allow users to perform corpus queries such as concordances and to generate frequency lists. However these tools lack support for larger billion word scale corpora.

Other server based tools exist such as Wmatrix [8], CQPweb [3], SketchEngine[6], KorAP [2] and corpus.byu[7]. Often these tools are based on Open Corpus Workbench (CWB)[8], existing relational DBMSs such as MySQL[9] or text indexers such as Lucene[10]. These systems handle corpora of larger scale better but are limited relative to the flexibility of local tools as linguists often cannot add their own corpora or annotation or are restricted in the size of corpora that can be added.

This lack of a clear solution or database management tool that can be utilized by linguists has lead to many systems being developed that are tailored for specific uses on specific projects as shown by Schneider [9] and Wills [10]. Much like with more generic server based systems the approach is often based on a relational database management system with a database design tailored to the needs of the project. More modern DBMSs such as MongoDB[11] allow for such systems to be built and scaled out (i.e. distributed across multiple servers) in order to allow a level of scalability. Distributed computing in general offers many solutions for processing larger text collections with tools such as Hadoop[12] and Spark[13] providing compelling options for parallel processing. However

[1] http://www.natcorp.ox.ac.uk/
[2] https://hansard.parliament.uk/
[3] http://eebo.chadwyck.com/home
[4] http://www.lexically.net/wordsmith/
[5] http://www.laurenceanthony.net/software/antconc/
[6] https://www.sketchengine.co.uk/
[7] http://corpus.byu.edu/
[8] http://cwb.sourceforge.net/
[9] https://www.mysql.com/
[10] https://lucene.apache.org/
[11] https://www.mongodb.com/
[12] http://hadoop.apache.org/
[13] http://spark.apache.org/

this form of processing is better suited to batch processing tasks such as tagging. Any use of such systems still relies on a degree of technical knowledge as many of these modern DBMSs and parallel computing software tools do not provide any direct support for corpus linguistic queries. Porta [7] has applied MapReduce to word counts, collocations and n-grams but through scaling up on one server rather than scaling out on multiple servers.

lexiDB seeks to fill the apparent void that exists between current corpus linguistic tools and more modern parallel computing approaches and DBMS solutions. lexiDB is a distributed DBMS designed specifically for storing and querying corpus data. It supports four corpus query types; concordances, collocations, clusters (n-grams) and frequency lists. It is designed to operate on static corpora - once a linguist has built a corpus they can add it to lexiDB and run queries. lexiDB can also be distributed across multiple systems to allow linguists to scale out as their corpus data scales up. This allows lexiDB to support corpora of the order of billions of words. Sections II and III describe the basic design and functionality of lexiDB and section IV presents performance figures for the tool.

II. Architecture

lexiDB's architecture is based on peer-to-peer (p2p) principles. All nodes within a distributed configuration maintain their own partitioned set of data. This p2p network of nodes is then accessed by a client. lexiDB can also operate with just a single node if distribution is not required with the client running on the same machine. The data on each node is organised into regions (similar to MongoDB chunks[14]). These regions maintain their own indexes of the data stored in them allowing for easy migration of data between nodes when new nodes are added. The size of these regions is user definable with different sizes yielding different performance benefits - typically larger regions are not recommended for systems with limited memory. Generally a region size of around 10 million words allows for a good balance of fast data reading and index build times whilst maintaining a reduced memory impact.

The index scheme utilised by lexiDB takes advantage of the Zipfian nature of natural language and as such uses a dictionary look-up system and stores occurrences of each word within a region as a single integer. Entries in this dictionary are unique based on the word in the corpus as well as any tagged information linked to it - i.e. if two words were textually identical but had different POS tags they would be stored as two separate entries in the dictionary. The dictionary stores all entries in lexical order, this allows for easier sorting of data when performing retrievals (e.g. value 001 is guaranteed to occur prior to value 003, alphabetically). On top of this a second level index is built which indexes the words stored in each region, in this way the top level index is much smaller and can be kept in memory and access to each region can be limited when performing a look-up to only access the regions necessary. lexiDB tends to be memory intensive - it is recommended that a system running lexiDB should have at least enough memory to hold an entire copy of the dictionary.

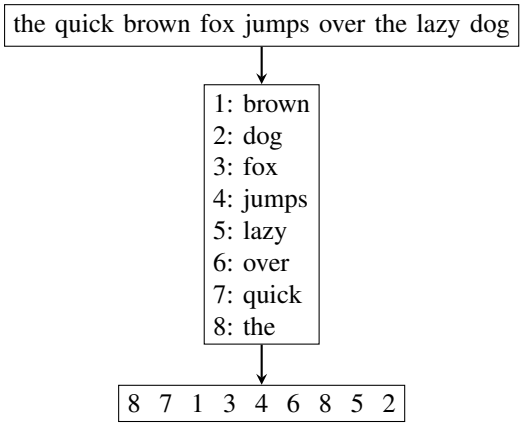

Fig. 1. Numeric Data Representation

Query execution in lexiDB resembles a map-reduce paradigm. A query from the client is sent to a particular server. The query is then distributed to all nodes or peers within the system configuration. The look-up is performed on each node using the multi-layered index described above and a result set is compiled on each node. The results are then returned to the server that initially received the query. The separate results from each node are compiled together on this one server before finally being returned to the client. lexiDB is designed to be agnostic of server / peer roles - i.e. there is no need for specific nodes to be designated as query handlers or data nodes which simplifies the configuration. All nodes should act as peers taking all roles. However lexiDBs configuration options do allow for such a separation of roles and concerns to be made if such a configuration is deemed beneficial or desirable - for example when nodes have varying hardware it may be beneficial to designate data nodes as machines with faster hard drives.

III. Functionality

lexiDB supports four corpus query types: concordances, collocations, clusters (n-grams) and frequency lists. Although generation of keyword lists is a typical task performed by corpus linguists it is not in itself a low data level query as it builds upon generating frequency lists from a target corpus and a reference corpus and using these lists to perform a comparative calculation of keyness. With varying methods of calculating keyness available (e.g. significance measures such as log-likelihood and chi-squared plus multiple effect size measures) the decision was made rather than to support generation of keyword lists to simply make generation of frequency lists as simple and efficient as possible to allow a user to then perform their own calculations for keyness in whatever way they see fit (as opposed to imposing a particular keyness measure). lexiDB clients can also provide this support directly.

As a proof of concept, we have created a vanilla command-line client for lexiDB rather than a graphical user interface

[14]https://docs.mongodb.com/manual/core/sharding-data-partitioning/

which could be added for demonstration purposes. The default client shell can be opened via the command:

```
> java -jar lexidb-client.jar [SERVER] [PORT]
```

where `[SERVER]` and `[PORT]` are the lexidb server (default `localhost:1289`).

A lexiDB server instance can be run using the command:

```
> java -jar lexidb-server.jar
```

A. Concordances

Concordances are created using the command:

```
lexidb> kwic [TERM/TAGS]...
```

Concordances or keyword-in-context (KWIC) searches (e.g. see figure 2) can be performed in lexiDB and searches support regular expressions (via automaton [6]). Tags for the search term can also be specified (tag search parameters may also be expressed as regular expressions). Queries can also be refined to limit the size of the result set returned. This is useful if you are searching for a very common word and are not interested in looking at thousands or millions of results. The size of the context window is limitless, although extremely large context windows on large result sets are not recommended unless the system has sufficient memory. Results may be sorted on words surrounding the keyword with two sort types. The first is a lexical sort (i.e. alphabetical), the second is a frequency sort which, for example, when performing a concordance search for "orange" and sorting by frequency one word before the keyword would return concordance lines with the most common word that precedes "orange" in the corpus. Sort orders can also be reversed. Finally, results may also be returned in the form of a file - this feature is particularly useful with large results sets as a linguist could then import the results into a spreadsheet and gather statistics regarding a vast set of concordance lines rather than just examining a handful of exemplars.

```
lexidb> kwic rural

3 concordance lines for "[rural]" retrieved in 2ms.

JJ XX II CC YCOM IO APPGE NN1 DDQ VV0 NP2 NN2 AT DA2 RG VM TO VBI IF AT1

 may be as commodious for  rural  Villages to be not incorporate
     in the Nature of our  rural  Deans , or of Archdeacons
the Number of the several  rural  Deaneries into which every Diocese
```

Fig. 2. Example of query results for concordance search

B. Collocations

Collocations can be created as follows:

```
lexidb> col [TERM/TAGS]...
```

Collocation searches support the same search term specifiers as in concordance lines (i.e. regular expressions and tag specifiers) but the searches return a frequency frame of words occurring within a specified context e.g. a search for "bath" with a context of two (default) will return all words that occur within that context and how frequently they occur in each position. Thus collocates can be examined with a single search and any calculations based on relative position can be applied without the need for further searches. Search results for collocates are sorted by total frequency of the word within the specified context.

C. Clusters (n-grams)

Clusters are created by running:

```
lexidb> ngram [TERM/TAGS]... [N]
```

Clusters or n-grams can be searched for in lexiDB using any value of 'n'. Unlike other systems that pre-build n-grams and store their frequency, lexiDB uses a similar context sensitive search to collocations and concordances to compute n-grams on the fly. Meaning rather than being limit to 4-gram or 5-gram searches lexiDB can support arbitrary values of n-grams. Searching, as before, supports regular expressions and the position within the n-gram of the search term can be specified or unspecified e.g. you may search for bigrams where the second word is "time". Search terms can also include specified tags to refine results. Results are always sorted by frequency but the ordering can be reversed.

D. Frequency Lists

Frequency lists are shown using the following:

```
lexidb> list [TERM/TAGS]...
```

Frequency lists can be generated using any form of regular expression. To view a frequency list for the whole corpus a regular expression matching all word types may be used. i.e. `.*`. Frequency lists may also be reversed in order.

IV. Scalability Results

To demonstrate the scalability of lexiDB, we conducted a series of test queries on one, two and four node distributed configurations using two, billion token scale corpora - Historical Hansard[2] (1.68 billion tokens) and EEBO[3] TCP phase 1 texts (0.91 billion tokens). Each of these corpora were tagged with lemma, POS, HT (Historical Thesaurus) and USAS semantic tags and stored in the form of `*.tsv` files. `kwic`, `col` and `ngram` queries were all performed using the ten most common words in the corpus (as found by the `list .*` command - results also shown). Each query was performed ten times and a mean query time was found. The most common words were chosen as this represents a worst case scenario for each query i.e. a query that has the most results to retrieve and it will likely be the most computationally and hard disk intensive and memory consumptive. All lexiDB instances were run on AWS[15] m3.2xlarge VMs (8 vCPUs, 30Gb RAM, 2 x 80Gb SSDs).

Figure 3 shows the time taken to insert and index each corpus on the three database configurations. The increase in speed is clearly evident as lexiDB is scaled out to multiple nodes, allowing for ever faster insertion times. This insertion operation is a one time process that a user of lexiDB would need to perform just once to execute queries against a static corpus. Whilst reasonably time consuming, it is significantly faster than we have experienced with other corpus systems, e.g. CQPweb takes over a day to index Hansard.

[15]https://aws.amazon.com/

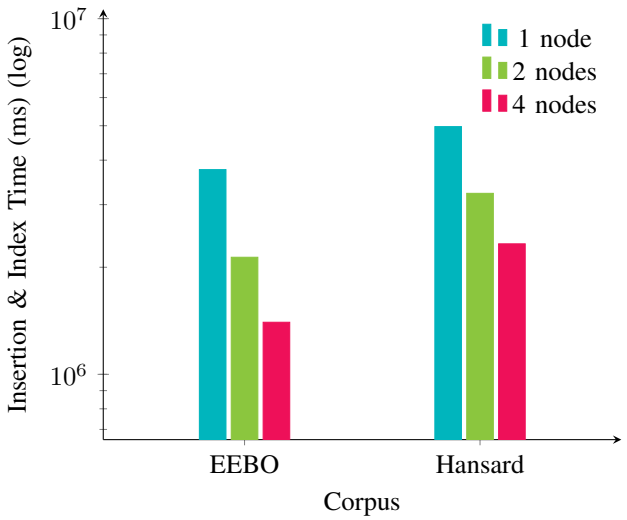

Fig. 3. Insertion and Indexing

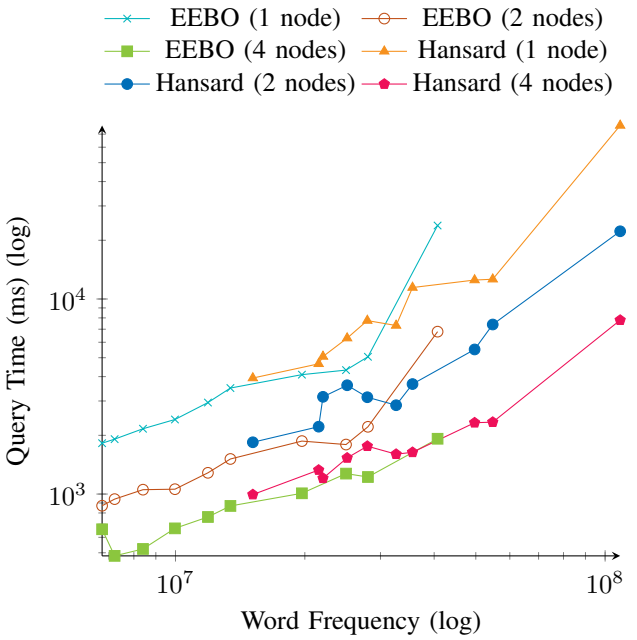

Fig. 4. Concordance Lines

The ten most common word types in each corpus used as search terms were {the, of, and, to, in, that, a, is, it, his} for EEBO and {the, of, to, that, and, in, a, I, is, not} for Hansard. Figures 4, 5 and 6 shows the result for the respective context sensitive queries (Concordances, Collocations and Clusters/N-Grams) using these lists of words. Because each corpus varies in size and the frequency of the words differs, the results are presented as the average query time against the frequency of the word type within the corpus.

Figure 4 shows the increased performance and reduction in query time for generating concordance lines. A concordance query for the word type "the" in the Hansard corpus which took 77,554ms on a single node was reduced to 22,242ms on a two node configuration and just 7,792ms on a four node cluster. This significant increase in speed is likely down to a reduction in the process time that will be utilized by the Java garbage collector while executing the query because retrieving the 108 million concordance lines returned from this query carries a significant memory overhead in lexiDB.

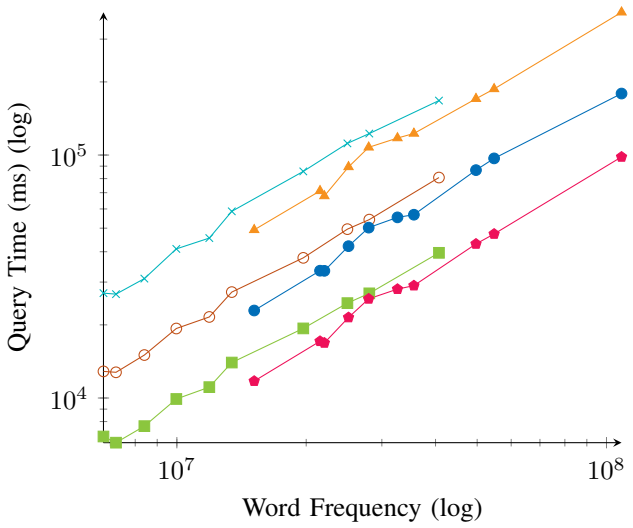

Fig. 5. Collocations

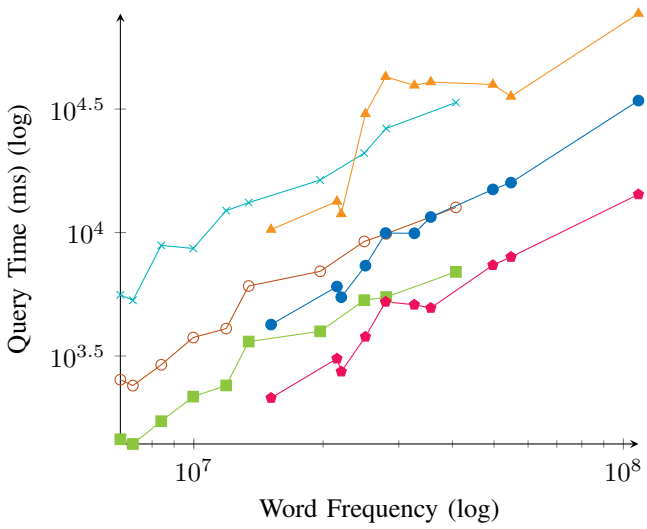

Fig. 6. Clusters (n-grams)

Collocation searches were run with a default context window of two words from the search term. Figure 5 shows the results of these. Again performance increases substantially as the configuration is scaled out from one to two to four nodes. The collocation queries themselves take far longer than the simple concordance queries due to the additional task of computing a statistics frame for the search term.

Figure 6 shows the results for cluster or n-gram searches. The query used was for a bigram of the specified search term. Similar to the results of the collocation queries, we see far

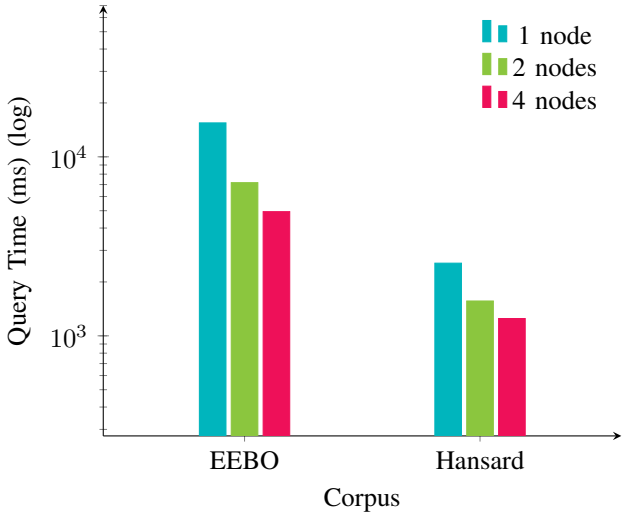

Fig. 7. Frequency List

longer query times than with a concordance search, again as a result of the computational expense of constructing n-grams from the retrieved tokens. This method of building n-grams on the fly, while more time consuming than pre-computing them at insertion time, saves significantly on disk space.

The time taken to generate the frequency lists are shown in figure 7. This was simply a list query for the regular expression `.*` which will return the frequency of all word types in the database. Obviously for this general case improvements could be made as the frequency lists for all words are pre-computed in lexiDB's dictionary and could be returned without the need to match a regular expression against the entire dictionary.

V. Conclusion and Further Work

In this paper, we have presented lexiDB, a new scalable corpus database management system designed specifically to support the indexing of text corpora and retrieval using the main methods employed in corpus linguistics. While other software achieves conceptually similar scalability, e.g. SketchEngine via virtualisation [4] and KorAP with Lucene/Solr integration, we believe that lexiDB is the first corpus database management system with in-built scalability via a distributed architecture. A key point to note about lexiDB is its fast data ingest time for extremely large scale annotated corpora. Normally this has to be traded off against fast retrieval time, but we believe that corpus linguists need both capabilities to deal with corpus updates in, for example, social network analysis where new data needs to be added regularly. Fast indexing also helps to reduce the time overhead between experiments, in other words if we improve the accuracy of automatic annotation and retag our corpus then we do not need to wait for 24 hours to complete the re-indexing before we can start obtaining updated results. lexiDB therefore addresses issues of 'velocity' in corpus databases and in turn, enables support for greater 'volume' and 'variety' in corpora indexed in the system.

We have demonstrated the capabilities of lexiDB through evaluation on two multiply-annotated corpora of the scale of one billion words and shown the extremely fast retrieval times for the most frequent words. In addition, due to the distributed design by adding more nodes, lexiDB is able to scale to even larger corpora and we will report on these results in further papers.

Future developments for lexiDB include adaption of the p2p architecture to create a SETI@home[16] style distribution network that can be used as an infrastructure by corpus linguists. This would enable lexiDB to be deployed and used, distributed, with billion plus word corpora for linguists who may not have the time, training or resources to deploy their own cluster configuration for the database.

lexiDB is open source and freely available on GitHub[17] under the GPL version 3 license[18].

Acknowledgements

This research is funded at Lancaster University by an EPSRC Doctoral Training Grant. Our research has benefitted from discussions within the Corpus Database Project (CDP) team including Laurence Anthony (Waseda University, Japan), Stephen Wattam, and John Vidler (Lancaster University, UK).

References

[1] M. Coole, P. Rayson, and J. Mariani. Scaling out for extreme scale corpus data. In *Proceedings of 2015 IEEE International Conference on Big Data*, pages 1643–1649. IEEE, 2015.

[2] N. Diewald, M. Hanl, E. Margaretha, J. Bingel, M. Kupietz, P. Banski, and A. Witt. KorAP architecture - diving in the deep sea of corpus data. In N. Calzolari, K. Choukri, T. Declerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, and S. Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 3586–3591, Paris, France, May 2016. European Language Resources Association (ELRA).

[3] A. Hardie. CQPweb - combining power, flexibility and usability in a corpus analysis tool. *International Journal of Corpus Linguistics*, 17(3):380–409, 2012.

[4] M. Jakubíček, P. Rychlý, and A. Kilgarriff. Effective corpus virtualization. In M. Kupietz, H. Biber, H. Lüngen, P. Bański, E. Breiteneder, K. Mörth, A. Witt, and J. Takhsha, editors, *Proceedings of the workshop on Challenges in the Management of Large Corpora (CMLC-2)*, pages 7–9, Reykjavik, 2014.

[5] H. Kucera and W. N. Francis. *Computational Analysis of Present-Day American English*. Brown University Press, 1967.

[6] A. Møller. dk.brics.automaton – finite-state automata and regular expressions for Java, 2010. `http://www.brics.dk/automaton/`.

[7] J. Porta. From several hundred million to some billion words: Scaling up a corpus indexer and a search engine with MapReduce. In M. Kupietz, H. Biber, H. Lüngen, P. Bański, E. Breiteneder, K. Mörth, A. Witt, and J. Takhsha, editors, *Proceedings of the workshop on Challenges in the Management of Large Corpora (CMLC-2)*, pages 25–29, 2014.

[8] P. Rayson. From key words to key semantic domains. *International Journal of Corpus Linguistics*, 13(4):519–549, 2008.

[9] R. Schneider. Evaluating DBMS-based access strategies to very large multi-layer corpora. In *Proceedings of the workshop on Challenges in the Management of Large Corpora (CMLC)*, 2012.

[10] T. Wills. Relational data modelling of textual corpora: The Skaldic Project and its extensions. *Digital Scholarship in the Humanities*, 30(2):294–313, 2015.

[16] http://setiathome.ssl.berkeley.edu/

[17] https://github.com/matthewcoole/lexidb

[18] https://www.gnu.org/licenses/gpl-3.0.en.html